ESD-TR-67-130

MTR-442

EVALUATION OF ADAM

AN ADVANCED DATA MANAGEMENT SYSTEM

R. A. J. Gildea

AUGUST 1967

Prepared for

DEPUTY FOR COMMAND SYSTEMS
COMPUTER AND DISPLAY DIVISION
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Project 512B

Prepared by

THE MITRE CORPORATION
Bedford, Massachusetts

Contract AF19(628)-5165

ESD-TR-67-130 MTR-442

EVALUATION OF ADAM
AN ADVANCED DATA MANAGEMENT SYSTEM

R. A. J. Gildea

AUGUST 1967

Prepared for
DEPUTY FOR COMMAND SYSTEMS
COMPUTER AND DISPLAY DIVISION
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Project 512B

Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts

Contract AF19(628)-5165

FOREWORD

This report was prepared under Contract No. AF 19(628)-5165 by The MITRE Corporation, Bedford, Massachusetts.

REVIEW AND APPROVAL

This technical report has been reviewed and is approved.

CHARLES A. LAUSTRUP, Colonel, USAF
Computer and Display Division

ABSTRACT

This report evaluates the ADAM project (Advanced Data Management System), its products, applications, and some of its activities, which were part of a larger project entitled Information Systems Tools and Software Techniques. The knowledge and conclusions contained herein are intended for Air Force and other personnel who either are systems programmers or have had a brief technical orientation in information processing systems, and are interested in the management and production of software tools. There are detailed evaluations of documentation and debugging facilities, system languages and language manipulators, data structures and memory allocators. Both the design and implementation of parts of the system, as well as the entire system are discussed.

PREFACE TO THE ADAM REPORT

This report was prepared under Contract No. AF 19(628)-5165. It evaluates the ADAM project (Advanced Data Management System), its products, applications, and some of its activities, which were part of a larger project entitled Information Systems Tools and Software Techniques. The knowledge and conclusions contained herein are intended for Air Force and MITRE personnel who either are systems programmers or have had a brief technical orientation in information processing systems, and are interested in the management and production of software tools.

The evaluation methodology employed is in form a three-step process, and in content reflects the considerations and opinions of experts on separate, but related topics. The initial step in the evaluation was the production of a very rough draft by seven of those who constructed and used ADAM. The second step was accomplished by having one person obtain additional information from interviews, and then reorganize and rewrite the draft. The third step in the process consisted of incorporating the criticisms and comments of eleven reviewers of the second draft. The ADAM System is a very sophisticated system and it was felt that a presentation of a tabular comparison with other systems would not yield as much information and evaluation as the presentation selected; i.e., evaluation of separate topics with brief explanatory notes about each.

The team approach that was adopted for the preparation of this report has the advantage of lessening bias from a single author and enhancing the possibility of a well-balanced report. The members of the MITRE technical staff who prepared this report in some cases had no prior connection with the ADAM project, and in other cases, were connected with the ADAM project as system programmer, design coordinator, user, or project leader.

TABLE OF CONTENTS

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONTINUED)

SECTION I

INTRODUCTION

In 1962 The MITRE Corporation, under contract with the Electronic Systems Division of the United States Air Force, started the research work on the design and construction of the Advanced Data Management System, ADAM. ADAM is a software tool to be used as a design aid for military information processing systems. It was intended that the ADAM concept would be validated by using it to design and exercise models in several application areas. At the conclusion of the ADAM project in August 1966, it had been used for modeling of five different systems.

The ADAM System is the most sophisticated general purpose data management system yet realized. The approaches and concepts used in ADAM have influenced the work in general purpose data management systems throughout the country. The amount of foresight of the personnel initiating and guiding the early design of ADAM should not be underestimated. The world of computer science five years ago is the environment for this foresight and not today's world with third generation computers.

The ADAM System is large and complex and is composed of both software and hardware. The hardware is the IBM 7030 (STRETCH) surrounded by a large disk memory, many magnetic tape stations, five query stations, and other peripheral equipment. Each of the five query stations has a line printer, an electric typewriter, and a cathode ray tube display console with push buttons.

Besides the three programs supplied with the 7030 computer that were used in ADAM (Master Control Program, MCP; STRETCH Macro Compiler, SMAC; and the STRETCH Assembly Program, STRAP), the ADAM software consists of 53 programs, with approximately 130,000 instructions (over 86,400 computer words) exclusive of large work areas. The majority of these programs constitute the main system which is run as a single job under MCP. The ancillary programs are run independently as separate jobs under the MCP. The main ADAM System may be operated with users not present, by use of card input, but the normal mode is with users on-line to the system via the query stations.

The objectives and a short history of the ADAM project are given in Sections II and III of this report. Section IV is the main section and is divided into four parts, each presenting the evaluation of a different aspect. The first part discusses how

many of the objectives of the project were met, while in the second part, ADAM is evaluated as a modeling tool from the experiences derived during the five applications. The construction of ADAM, covering some of its features and the programming environment during construction, is discussed in the third part. Consideration of some important problems outside the scope of the ADAM project is given in the fourth part of Section IV. Section V contains some conclusions derived from this research project that may be of interest to designers and builders of future general purpose data management systems. In the Appendix is a bibliographic list of documents pertinent to the design, construction, use, and applications of the ADAM System.

SECTION II

TECHNICAL OBJECTIVES AND PLANS OF THE ADAM PROJECT

The research problems to be solved in implementing the ADAM program were substantial. Viewed with respect to then-existing achievements of the state-of-the-art, the list of objectives reflects this.

The Electronic Systems Division of the United States Air Force and The MITRE Corporation initiated activities on the ADAM (Advanced DAta Management) System as an experiment in creating and developing special tools and design aids for systems which process military information. Large volume and complex relationships characterize the data of a command data base, while a rapid response and increasing requirements characterize the procedures of the information processing systems. The enormity and complexity of understanding enough to specify and formulate an integrated and balanced system in a timely manner renders pencil-and-paper techniques unfeasible. The need for computer-aided design is inevitable in military information processing systems.

Some of the following information about ADAM objectives and plans has been taken from pertinent ADAM planning documents* and appears within quotation marks. Some of the objectives herein are taken from the sections called Project Work Plans. In a few cases there has been slight editing of the quotations for improved readability, especially in the verb tenses.

"The Air Force Electronic Systems Division needs a constantly improving capability to plan, design, and evaluate command and control systems. This capability must include the best available methods for generating alternative system designs and precise techniques for rapidly evaluating existing, prototype, or proposed system designs or design characteristics.

"In particular, with respect to information processing, system designers must be provided both with improved operational concepts of information processing and with improved techniques and flexible tools for use in the design and evaluation process. These tools and techniques must be capable of rapidly reflecting the latest known technology and the latest experimental or proposed designs."

---

*ESD/MITRE Technical Objectives and Plans, Information System Tools and Design Techniques, Project 502, 1 August 1963 and 15 August 1964.

The primary intent of the ADAM project was to provide for the system designer a laboratory facility for his design work. The secondary intent was to continue research in information processing. Because of the research nature of the project, many of the objectives initially set forth proved to be more fruitful than others, hence modifications and amendments to the list of objectives occurred. The intent to validate the ADAM concept for field use was also important from time to time in the life of the project.

The ADAM System

"The objectives of this project were to investigate, develop, and evaluate advanced information processing techniques for use in the system design process and to make available rapid means for realizing these techniques in an experimental setting." In order to realize these intentions, the objectives of the ADAM System task were delineated as follows:

"Develop and implement a computer-program framework into which potential applications of new computer processes and procedures can be inserted rapidly and evaluated precisely."

This program was to be adapted to specific simulations and have the following capabilities:

1. "Generation of user files:

    a. from almost any machine-sensible form of the data as a source;

    b. by subsetting existing files;

    c. by restructuring existing files.

2. "On-line updating of ADAM files by changing values, adding new objects, adding new properties, adding synonyms for any name, and deleting parts of files.

3. "Provision for the inclusion of codes specific to an application system. Provision was to be made to allow programming of these routines in a higher-order language. Compiled code was to be 'called' by the ADAM System by means of the operator language. Data and storage requirements of the routine were to be relatively impervious to data-base changes and were to continue to operate properly as long as the elements of data required by the routine continued to exist in the data base.

4. "Provision for a basic user language within ADAM for on-line insertion of user requests. The translation rules for the operator language were to be externally specified and processed outside the translator for inclusion in the translator. Because of the general techniques to be used in this area, it had to be possible for the user to describe a new language and have ADAM understand it rapidly. In addition, definition of abbreviations, synonyms, strings, stored queries, and formats were to be accomplished on-line.

5. "Utilization of the input-output stations provided in the Systems Design Laboratory for the 7030 computer. These were to be dynamically configured by the system as required by the test director, and appropriate legality checks were to be made.

6. "Generation of reports (and query responses) conditional on events occurring in the experiment. Information in the reports was to be organizable according to operator specifications, including the preparation of summaries and sorting on multiple keys. The display format for reports (or query responses) was to be specifiable by the operator and possibly varied by him dynamically."

SECTION III

HISTORY OF THE ADAM EXPERIMENT

In the spring of 1961, programming activities began in support of the MITRE work on an experimental facility for transport planning and modeling, known as the Experimental Transport Facility (ETF). Fortunately, the ETF program was modeled, designed, and implemented, especially in its externally-described data base structure features, in such a way as to allow its application to some of the problems in Project 481, Post Attack Command and Control System (PACCS).

Several successful demonstrations of ETF were conducted in 1962 and 1963, wherein remote teletypewriters were connected through long-distance telephone lines to the XD-1 computer and a "high-speed" printer. By extending some of the design and implementation features, broader application of the ETF approach could be made. Experience with the transport planning function and the Project 481 application indicated that the generalized approach used in ETF was valid. By modifying these approaches and adding new concepts, ideas for an extremely advanced system evolved. Many of these concepts were new while others were taken from the more advanced technology in information processing.

In August of 1962, work was initiated on the ADAM project to provide at least the same ETF capability, but significantly enhanced from the point of view of the user in an experimentation facility. At this time, design concepts were established for the data structures of the system, and by fall, overall system design was reflected in functional block diagrams of the major program components.

The supporting 7030 software provided by the vendor was inadequate as a programming environment for the specialized needs unique to development of a large data management system. Because it was felt that programming productivity was inhibited, certain remedial steps were taken, including the revision of the macro compiler and the development of specialized production aids.

The three major design decisions that occurred during March and April of 1963 were to employ a syntax-driven translator for the query language, to implement an interim version of the system compiler, and to revise completely the file generation approach.

By August, some more internal formats of data structures were defined. Functional requirements of other corporate projects influenced the design of the first experimental ADAM query language, FABLE.

Before the end of calendar year 1963, work started on designing and implementing the display scope and light pen input capability. This non-typewriter input capability, integrated into the ADAM System during the following year, thus enhanced man/machine interface for query language operations. Also in the fall of 1963, the decision was made to have output information placed in the format of files rather than in tables. Provisions were made during the following January and February to allow FORTRAN subroutines (with certain limitations) to operate within the ADAM System.

A significant amount of work was discarded when the IBM 7750 input-output computer was deleted from the hardware system configuration early in 1964.

A reduction in force on the ADAM project occurred during February and March and it was difficult to minimize its effects on the ADAM development effort.

In attempting to solve some of the persistent difficulties in file generation, a third approach began in the summer of 1964 by merging file generation and retrieval. This proceeded so that in December the parts of the ADAM System were sufficiently developed to allow operation in responding to its first query. The system became available to users in February 1965.

During 1965 increased capability was given to the experimental query language, testing individual program components was continued, and the difficulties of matching the interfaces between modules were isolated and removed one by one. As more of the system became operational, execution times were measurable.

In the spring of 1966 the file generation was once more investigated; although it was functional, it was considered too slow. By reorganizing the program, most of the previous work was saved and execution time was cut to one-third.

To eliminate the confusion that sometimes occurred when many users were operating the system simultaneously from different consoles, in the summer of 1966 every input message became separately identifiable.

Although work on the project officially ended on August 31, 1966, several exercises have occurred since, including a demonstration of ADAM given on December 27, 1966.

The ADAM System was applied to experiments in data management in five areas, and involved a period of activity from January 1965 to December 1966.

Work in one of these areas, the Joint Air Force Logistics Command/ESD/MITRE Advanced Data Management (ADAM) Experiment, took place from about January 1965 to August 31, 1966. Use of the AFLC ADAM System began in April 1966 by operating the ADAM System on the IBM 7030 computer at Hanscom Field in Bedford with a single remote console at Wright-Patterson AFB, Dayton, Ohio.

The Man-Job-Match (MJM) project of the Computer-Centered Data-Base Systems effort of ESD started in January 1965 with its statement of work and continued to the end of the calendar year 1966, beyond the official close of the ADAM project in August 1966.

In February 1965 an ADAM application called the Satellite Test Center was started, and in November of the same year, its demonstration was given.

The Manned Orbiting Laboratory (MOL) work occurred from April to July 1965 and included several demonstrations of the MOL ADAM System early in July 1965. Further support to MOL Activities were continued until March of 1966.

Work on the Tactical Airborne Beacon System (TABS) occurred from January 1965 to October 1965. Several hours of successful operation of the TABS ADAM System were logged during October 1965.

SECTION IV

EVALUATION

A. REALIZATION OF EXPLICIT OBJECTIVES

The intent of the ADAM project was threefold: to produce a laboratory tool for computer-aided system design, to conduct programming research, and to validate the ADAM concept for field use. The use of the ADAM System in applications to five widely varying information processing systems attests to the fact that ADAM was used operationally as a laboratory design tool. Further discussion of these applications are given in the section entitled MODELING. In particular, the discussion of the AFLC experiment contains some information on the remote operation of ADAM by mission-oriented personnel in the field.

The programming research performed in solving some of the problems of general purpose data management has allowed The MITRE Corporation and ESD to maintain their position in the field of information processing. Only a small fraction of the total technical knowledge derived from this research has been documented; much of it is being carried over into other projects.

A summarized list of the detailed objectives appearing in Section II, TECHNICAL OBJECTIVES AND PLANS OF THE ADAM PROJECT, is given below.

Summary of Explicit Objectives

1. File generation
   a. from cards, magnetic tape, et al;
   b. by subsetting;
   c. by restructuring.
2. On-line updating of files by
   a. changing values;
   b. adding objects;
   c. adding properties;
   d. adding synonyms;
   e. deleting.

3. User Routines

4. Query Language

5. I/O Station Selection

6. Event Triggered Reports

File generation with the ADAM System can be accomplished by having the new file generated either with data from some external storage medium as input or by processing data that already exists in previously created internal data structures. Subsetting a file already existing within ADAM to form a new file was the usual activity in most of the applications of ADAM, and the ability to restructure a file was used to great advantage, especially in the MOL application.

Although the ADAM System could accept information from many input media and it was possible to generate files by subsetting from an external medium, it would have taken so much time for the 170 million character data base of the AFLC experiment that it was not a reasonable approach in this application.

The on-line capability for the ADAM System was provided in two ways: typewriter input and display console (non-typewriter) input. Queries written in the experimental language, FABLE, not only allowed for processing and retrieving data from files but also for changing values, names, and descriptions of individual properties or files. New objects, or values of repeating groups, could be added or deleted with relative ease. Deleting properties or adding new ones to files was accomplished by creating a new file with the desired configuration. The supplementary capabilities of the utility language and string substitution mechanism, which are described later, were found to be extremely valuable and were used to a great extent both on-line, with either typewriter or non-typewriter input, and off-line with card input.

The ADAM concept provides for a flexible and competent data management system as a basic system, but also recognizes that additional unique capabilities will be required by each application. The ADAM System allows the assimilation of a user's routines for his own specific application by storing them in the Routine File along with the system routines. Although the user routines were written in all of the user languages, they were stored in relocatable machine language in the Routine File. Exceptions to this are found in the non-typewriter input capabilities, for which skeletons of source language input queries and messages are stored in data files.

The experimental query language, FABLE, most often provided the means by which users controlled the particular activities of the ADAM System for their applications. Provision was made in the query language to allow for appeals to any of the system or user routines stored in the Routine File. As the technical objectives demanded, the experimental languages, IFGL, and three versions of FABLE, were defined in a compiler-building language and all were recognizable by a single ADAM table-driven translator. Both these query languages and the string substitution mechanism of ADAM allowed a great deal of freedom to the user in naming, renaming, and establishing synonyms and abbreviations. More detailed discussion and evaluation of language capabilities occur later in this report. The objectives for the functions of creating, changing, and manipulating data and files, as well as the objectives concerning user language capabilities, were not only met but, in many cases, alternative approaches for accomplishing the same function were made available to the user.

Punched cards and magnetic tape were the customary input media especially for large amounts of data, but the list of input media also includes electric typewriters, push buttons and light pen from the display scope, local and remote teletypewriter, and equipment for receiving real-time information. It must be noted that the generality of the input section of the ADAM System allowed for the addition of several of these input equipments during the ADAM project with only slight modification to the ADAM program and the Master Control Program for the 7030 computer. The need for accepting information from still more devices was not evidenced during the ADAM project, but it is reasonable to expect that accommodating such new devices would probably produce no greater difficulty than the others. ADAM was constructed in such a way that all query output messages received uniform treatment until very late in the throughput sequence. A single formatting routine formatted all output messages by means of tables of characteristics of output devices, and then transmitted the messages to the outside world. Prior to the formatting, any message could go to any output device; the selection was at the user's option.

Within the objective "Event Triggered Report," an event was intended by the ADAM designers to mean the occurrence of a change in a property value or the elapse of a specifiable amount of time. The implementation of such event detection in ADAM was not completed during the project. The "Event Triggered Report" objective also involved general requirements concerning all reports, independent of their cause. These specifications were completely realized, so that query responses can be organized, sorted, formatted, and transmitted to dynamically selectable output devices.

The objectives of this list are not of equal value, nor do they constitute a comprehensive list of all the ADAM project objectives, because many others were imposed that did not appear in the project documentation and/or were indistinguishable from design specifications.

Perhaps most striking in the examination of the ADAM System is the realization that the original objectives of the ADAM project are not only very ambitious, but that many of them appear relatively nebulous.

B. MODELING

The building of large information processing systems is very costly, as is the building of prototypes of them. Quite often, the cost and lead time for designing, building, and changing the software significantly exceed those of the hardware. By providing the ADAM System as an experimental design tool it was intended that the design of information processing systems would be influenced for the better. Betterment in this sense may be in the form of:

1. reducing the length of time for the design phase;

2. more complete intial design, that is, without logical gaps;

3. more consistent initial design, that is, with fewer errors and all the subsystems specified to commensurate levels of detail;

4. greater capacity for changing the initial techniques by improving them or substituting new techniques;

5. greater capacity for adding new capabilities and retiring old ones gracefully.

In order to examine the behavior of a proposed design, a model is often built and exercised. The level of abstraction in the model depend on the goals of the exercise, the type and capability of experimental tools, the capability for validation of the model, and of course, the judgment of the investigator. For a large and complex system, building and exercising a model that is detailed enough to be a prototype was felt to have far less probability of success than building and exercising a small prototype system. Each information processing system has some time-consuming executive and housekeeping operations superimposed on its capabilities as overhead time. To impose additionally the overhead of a very flexible experimental design aid upon an already detailed model can easily compound, and thus extend response time of the system beyond desirable or acceptable limits. Inasmuch as the goal of the modeling is to demonstrate feasibility, it is unwarranted to strive for the inclusion of all capabilities of a full system.

The AFLC/ESD/MITRE Application

The application of ADAM to the Joint Air Force Logistics Command and Electronic Systems Division experiment involved user operation of the ADAM System on the 7030 computer at Hanscom Field, Massachusetts, via a query station located at Wright-Patterson AFB in Ohio. The data

base consisted of information about a number of subsets of items, each numbering from 1000 to 3500 out of the 85,000 items in the full inventory. This project included not only experiments using the data base, but also the functional duplication of the computational capabilities of an operational system. Data base investigations were done within minutes by having the analyst on-line with the data base via ADAM. The data base was investigated by queries directed towards searching out salient characteristics of the data.

During this joint experiment, two separate specifications were provided for these computations, which are usually performed quarter-annually. Likewise, the data content and format of the 64 reels of master file magnetic tapes had two versions. The modeling for both versions of the computations was accomplished completely in less than six man-months. The same ease of use was not provided by ADAM to the modeler in initially subsetting his external data base. Each AFLC data base subset was instead obtained by passing the data from the 64 reels of master file information through many FORTRAN pre-processing programs, and then through an ADAM file generation procedure. This difficulty indicates deficiencies in the ADAM experimental file generation language or its implementation because its use for this subsetting function would have consumed approximately 20 times as much computer time as the approach taken by applications programmers.

Despite some of these difficulties in subsetting, and although the full 85,000 items could not be practically handled, it would be misleading to indicate that only 3500 items were available. In fact, eight subsets were formed and employed in two ways, but in neither case was simultaneous processing done on more than one subset. In one case, many procedures were operated against one subset arbitrarily selected prior to run time. For the other, different reports were generated by running the same procedures against several subsets, one after another within a single MCP job. The capability for substituting data subsets that are themselve complete and compatible data bases into the whole system without damaging their environment is itself a breakthrough. The breakthrough was possible because the data descriptions were wholly contained within the data base for these file structures. Thus once the ADAM-based system for the experiment had been created, the changes to it to treat new data subsets and computations were nearly trivial. The users commented that the ADAM data structures should have allowed either an array as a subfile structure, or else random access to any value in a repeating group. Sometimes extra coding was introduced into a procedure because the design of the repeating groups included only capability for serial processing.

The Man-Job-Match Application

One of the ADAM applications, called "Man-Job-Match," is itself a small prototype system. It could be considered one capability in a larger personnel system of the future, if such were to be developed. By processing a 5 million character data base with selected queries, the system extracts the available manpower with proper experience and qualifications to match open jobs, storing men and jobs in separate working files. Essentially, there are two main files, the job file with approximately 5800 jobs listed, and a man file with the same number of men. Approximately 120 properties are used to describe each job, while approximately 150 properties are required for each man.

After further screening is performed by qualifying every man against each job in turn, a large system of FORTRAN routines compatible with ADAM, completes the matching of men against jobs. Using the Hungarian algorithm of linear programming, men are then assigned to jobs. This prototype system is sensitive to on-line action reflecting policy changes for both availability and evaluation of men.

A file of about 75 queries is stored in the system. Queries from this file are selected and qualified via light pen and push buttons. In addition to these two input media, the full query capability of the ADAM System is available via typewriter or other input mechanisms.

In many experiments and demonstrations of this application, the display language capability enhanced console operations for the user. It can also be exercised running in batch mode operation of the 7030 computer, that is, using punched card input rather than console input.

The modelers for this application commented that although the FORTRAN capability was a necessity for practical considerations, the interface between FORTRAN and ADAM routines for both control and data was awkward because of the detailed clerical requirements that the programmer, rather than the system, had to manage when writing his interface routines. Thus the compatibility was not achieved in an optimum manner.

The DAMSEL compiler and the Man-Job-Match application were being developed in parallel, and the application programmers were often frustrated by the faulty output code of the incomplete and unchecked compiler.

An interesting consideration of the Man-Job-Match application of ADAM is that its model has some very realistic characteristics. For example, approximately 30 men and 30 jobs can be matched in about 30 minutes. It is not unreasonable to expect that the needs of one of the military commands could be met by running such a program several times a month. The Man-Job-Match application is thus a good one to use to support the thesis that the ADAM concept is feasible. It required in its model the manipulation of a data base of a few million characters and extensive numerical computations, and the model ran at reasonable speed.

Satellite Test Center Application

The Satellite Test Center application was a MITRE sponsored experiment using ADAM. The model contained the capability of displaying the schedules for up to 20 observation sites each on as many as 13 satellites. Along with typewriter and line printer output, the system utilized the display scope for showing when the satellites could be observed from each of the sites. For example, a typical display would represent the information for one site, and be organized so that a row was given for each satellite, and one column to each hour of the day. Whenever this site may observe a given satellite in the row corresponding to the satellite, a horizontal line segment is drawn through those columns representing the time period of observation. From this start, there was triggered a considerable amount of interesting analysis and programming on scheduling policies, conflict resolution, and associated specialized display techniques. Although the demonstration of this system was given in November of 1965, much of what was learned on this project influenced the activities on both the ADAM programming effort for the Manned Orbiting Laboratory and the subsequent ADAM MOL support work. Many of the Satellite Test Center routines were based on the routines for the display language capability of ADAM and were themselves used in the MOL support activities. An interesting capability that seems to hold a lot of promise as a general planning tool is the one that allows a user at a display scope to re-position line segments of the display by means of a light pen. Displays could be made up and modified easily using the light pen. They could show in graphical form when activities would be occurring, such as in the manner of a Gant chart.

The Manned Orbiting Laboratory Application

The Manned Orbiting Laboratory application of ADAM extended over a period of less than four months. During this period, the model was designed, implemented, and exercised. During the work, the model was redesigned and as a result, several files had to be restructured. The ADAM System is most helpful to a modeler under these circumstances;

that is, when the total problem and the data itself do not change, but the processing required and the structure of the files do change.

It was estimated by one of the modelers that without the use of the ADAM System and using conventional programming techniques, the design and implementation of the model would have taken at least eight times as much effort as it did using ADAM.

The ADAM System served well in the MOL experiment and high-lighted some of the advantages of the general purpose data management system approach. The biggest difficulty encountered in the MOL experiment was the creation of the large coverage file which kept the observation data for the many passes of the satellite over each station. In the opinion of the application programmers, the greatest advantages of the use of ADAM for this application were the ability to restructure the files with FABLE and the short response time obtainable in on-line use. The value of being able to restructure files on-line and test the new file design immediately was appreciated early in the project.

The approach of storing a good deal of redundant data in the coverage file proved to be costly because it lengthened the processing time.

Some of the functions performed involved the use of light pen and display scope.

This work exposed several problems in scheduling and conflict resolution that were the subject of investigations in the MOL support activity that followed, and the Satellite Test Center project. The MOL support work borrowed the problem and data base from the MOL project, and the display techniques and capabilities from the Satellite Test Center project.

The Tactical Airborne Beacon System Application

The Tactical Airborne Beacon System (TABS) was also a prototype system. Position information on aircraft was received in the form of beacon signals by radio frequency receivers. The signals were processed only slightly and transmitted by communication equipment to the 7030 computer. Occasionally, command signals from display and control panels were sent to the 7030 computer. The position information was fed to tracking programs and upon conclusion of the calculation, status and history files were updated. The command signals, when received by the ADAM System, activated pre-stored queries that transmitted to the display panel the desired information about any or all of the aircraft being observed.

The modeling using ADAM proceeded without much redesign of ADAM routines. Probably this was due to the fact that required data and file volume transmission rates and processing were not so severe as to require performance beyond ADAM's capabilities.

When a general purpose data management system is being used as a tool to model a given system, the model is usually affected and shaped by the capabilities of the tool. The TABS application model was least so affected of the five ADAM applications. In the ADAM experience, the impressive observation drawn from the TABS application is that ADAM was able to operate on-line with the TABS equipment in real time despite the vast overhead of this pioneer general purpose data management system.

Summary

Although the modeling and remodeling with the ADAM System for the above five projects were accomplished fairly easily, it must be noted that some of the models and their data bases were adjusted to fit within the ADAM System. In the case of the AFLC project, the operational data base is approximately 25 times larger than the one used in the ADAM System experiment. The Man-Job-Match project used as its data base the data (as of December 1965) for the 5800 men and 5800 jobs comprising the Weather Career Field and Personnel Career Field of the Air Force.

In many cases, there is little advantage to be gained by experimenting with the full data base, yet there are other cases for which this is true only initially; as the experimentation develops, more and more of the full operational data base must be employed for a true picture of the system.

It was noticed that, in general, the greatest aid that ADAM provides to a modeler is in investigating the problem to determine requirements. If the details of the total process, e.g., computations, formats, and structures of data, have not yet been determined, ADAM provides a rather flexible and useful tool for experimental formation of the data base and procedures, and design of output formats.

In using ADAM to imitate another system that is already operating or specified, much less freedom is provided to the modeler. Of course, ADAM cannot model any arbitrary system. It models those systems best which:

1. allow adequate representation of their data bases in the ADAM serial file structure and in about six million characters or less;

2. benefit from the use of a generalized translation or encoding mechanism, i.e., the ADAM roll which permits encoding arbitrary length names and data values into more facilely manipulated fixed length form;

3. require an on-line query capability consisting mainly of single file subsetting, simple file updates, and limited cross-file referencing;

4. do not require a general on-line computational and procedural programming capability, but are satisfied with such a capability off-line;

5. may require local or remote display, light pen, typewriter, and line printer capabilities;

6. benefit from a special-purpose language implemented with a modest investment;

7. may require specialized routines and thus benefit from the convenient assimilation of such coding by the ADAM System;

8. have no severe real-time constraints;

9. place a premium on reduced lead time for system definition and evolution;

10. benefit from format specification which can be independent of specific files;

11. have not been fully specified, and for which some degree of freedom remains in the determination of requirements.

C. ADAM DESIGN AND IMPLEMENTATION

1.0 Documentation Facilities

In his article "Management Techniques for Real Time Computer Programming," T. Holdiman states: "...the craftsmanship of the programmer cannot be judged by examining his piece in isolation but rather only as it supports and interacts with other pieces."* A key element in the successful integration of one program module with the other modules in the system is the set of documentation facilities and conventions that are part of the design and production environment.

Initially all ADAM papers were in the form of APS (ADAM Project Serial) documents. Two types of documents existed: relatively official documents (designs, specs., etc.) and relatively unofficial documents (proposals, ideas, etc.). The official documents were produced in the name of one of the ADAM groups, while the unofficial documents were produced in the name of the author. APS documents were maintained by numerical order in a master notebook in the ADAM Administrative Office. They were distributed to persons concerned according to a set distribution scheme. As the numbers of APS documents increased, they were organized into logical categories in order to make the whole series more readable. Also, a second important change was implemented. A new series of documents, the D-Spec series, was issued. The D-Spec, or Design Specification, document was a firm specification for the design of the ADAM System. Each series was produced and maintained separately according to a definite set of rules. The D-Spec papers were considered official ADAM documents while the APS group was a relatively unofficial series which did not cover design specifications.

As system debugging progressed, three new document series were initiated: ADAM Newsletters, ADAM Flashes, and Deck and Tape Notices. ADAM Newsletters presented informal, timely information concerning system use, debugging, and integration. ADAM Flashes required only one hour from writing to distribution and were used to report catastrophic errors, etc. Deck and Tape Notices reported the status of the latest system decks and tapes.

When the ADAM System became operational, three additional document series came into existence: Production ADAM System Releases, Production ADAM System Newsletters, and Production ADAM System Flashes.

Eventually many important documents from all series were used as the basis for the ADAM User's Guide which was published as a MITRE Technical Report, MTR-268.

---

*T. Holdiman, "Management Techniques for Real Time Computer Programming," Journal of the ACM, Vol. 9, No. 3 (July 1962).

Our evaluation of the documentation facilities provided is that they were well designed. It is unfortunate that during design and coding they were used only in a passive way. It was common practice during the construction of ADAM for the documentation to be produced only after the programming had been accomplished.

Experience in the ADAM project indicates that for such a large programming effort there is an inherent requirement for training, despite the absence of plans in the project outline for formal training of users. The system programmer in one section is often the user of other parts of the system and has critical need for the documentation. Prescinding from the many good features of the ADAM documentation, two difficulties can be noted for the benefit of future system builders.

It was found to be very frustrating to many programmers in search of specific information to have to read document after document, picking up in each only a fragment of the desired information and getting a reference to another document for further details. Moreover, updating such inter-referenced documents becomes exceedingly time consuming. The other point to be noted is that although the desirability of documentation was known, there are, nevertheless, parts of the system for which no documentation exists. Much of this can be ascribed to conditions that sometimes overlapped. Our experience was that when schedules for producing programs were jeopardized or slipped, the first thing affected or completely set aside was the documentation in order to make more room for programming. It is also not uncommon to find that the extremely talented, technically oriented people are not very proficient at expressing thoughts in written form.

2.0 Debugging Facilities and Production Procedures

Introduction

The creation of a system as large as ADAM produces problems in debugging and production of a magnitude much larger proportionately than experienced with small programs. This is one of the reasons for the unfortunate fact that the cost per instruction of a program increases with the size of the program.* It is easy to see why this is so. Time does not permit all the components of a large program to be produced serially, and instead they must be produced concurrently. This concurrency of production, undertaken in present day environments,

---

*D. F. Parkhill, The Challenge of the Computer Utility, Addison Wesley, 1966.

produces a degree of interference between undebugged programs which boggles the imagination.* The creation of a satisfactory system of programs as large as ADAM not only justifies, but requires the development of system-wide debugging facilities and production procedures.

It should be recognized that the support software provided with the IBM 7030 was in no sense adequate for the development of a large program. Efforts by The MITRE Corporation to modify, extend, and otherwise enhance the basic capabilities of the operating system and support software were carried on, and were required by the ADAM System in order to proceed at all. The resultant system should not be considered as exemplary of support mechanisms specifically intended to support large programs.

Description of ADAM Debugging Facilities

At the conclusion of its development, the ADAM System included the following debugging facilities:

1. SINTAB Trace - Most queries processed by the system in either IFGL or FABLE languages are operated interpretively as a set of SINTAB entries. This facility is turned on by setting a bit in core. It prints out the address of each SINTAB entry as it is executed and the elapsed time.

2. Logging - This facility allows certain programs in the system to log the time and a preset message on a system output tape.

3. Routine Trace - This facility allows routines to be selected for tracing. Upon entry to the routine, some fixed information can be printed on a system output tape.

4. COREMAP - Either upon the explicit direction of a program or under certain abnormal conditions, an annotated map of core storage can be printed. This enables memory dumps to be properly interpreted despite the context of dynamic storage allocation.

5. SINDUMP - This is a mechanism which prints SINTAB in an easily read format.

---

* T. Holdiman, "Management Techniques for Real Time Computer Programming," Journal of the ACM, Vol. 9, No. 3 (July 1962).

6. Translator Flow Trace - This mechanism logs on a system output tape the execution flow through the interpretive diagrams of the translator, and the input to individual generators.

7. Relocatable Cards - These cards allow the correction of binary versions of relocatable programs.

8. I/O Logging - This includes logging of all input as received from input devices and is necessary to determine errors in hardware.

9. Trap - A trap can be set to produce a coredump following the execution of a specified routine. The trap is set through a message preceding the message for which trapping is desired.

10. Many other special purpose and miscellaneous facilities - MC, POINTS, $TOPLOG, $ON, $OFF, etc.

Evaluation of ADAM Debugging Facilities

As can be seen from the above list, the set of debugging tools provided by ADAM is quite extensive. However, despite this investment in debugging facilities, ADAM was a very difficult system to debug, in the opinion of a majority of the programmers who worked on the project. This difficulty reduced the productivity of the programmers and thus affected schedules and costs. Listed below are some suggestions by which the conditions producing the difficulties during ADAM production can be ameliorated for the debugging of future systems.

1. When selecting a computing system, i.e., hardware and software, consider how well exercised and general purpose in nature its debugging facilities are because of the impact in usefulness and reliability.

2. Include in the system design, initial debugging facilities with approaches and conventions for subsequent add-on facilities. Thus, there can be an integrated set of debugging tools, rather than a collection of ad hoc routines.

3. Provide some automatic method by which the operation of a new and old version of the system may be checked for consistency.

4. Provide for the programmer the option of debugging on-line or off-line. This probably implies a time-sharing facility for large computing systems with high cost.

5. Provide computing services to the programming staff with turn-around time of less than one hour exclusive of job running time.

6. Consider readability of the program when coding. At the start of the ADAM project, programmers were not sufficiently discouraged from considering the complex timing of individual IBM 7030 instructions. This has historically been one of the programmer's most interesting roles, but the timing considerations of the IBM 7030 are so complicated that such considerations occasionally led to artificial and obscure coding techniques.

7. For the purpose of early system integration, internal communication, and maintenance efforts, consider the advisability of conservative selection of a subset of the computer instruction repertoire. The entire ADAM System was coded in relatively unrestricted machine language. Coding the system in either a higher-level language or restricted machine language could have prevented the problems caused by wild branches, wild stores, etc. No appropriate high-level language other than FORTRAN was available, and the IBM-supplied macro facility was not sufficiently debugged.

8. Strive to simplify and standardize control and data interfaces. The design of the interfaces between the user and the roll routines, allocation routines, and data structure primitives encouraged a great number of errors because they were too complex.

9. Be alert to the fact that some system difficulties are engendered when single routines are allowed to be very large. The large size of the ADAM program modules prevented interroutine tracing mechanisms from being effective.

It is thus clear from our experience that even the best programmers, when building a large program, require fast turn-around on a reliable computer enabled by adequate software, and also a work discipline that is imposed because of the size of the system. Each programmer when working alone disciplines himself by following his own design decisions, coding conventions, and debugging procedures, which most often are not written down. When the programming effort is large enough to require many programmers, then these things must be written down so that all

the programmers follow the same discipline. This discipline affects the initial design, the implementation, and even as it did in the case of ADAM, the actual selection of the machine instructions.

The purpose of recording the experience in developing ADAM and in evaluating the experiences is to benefit future activities. The individual programmer can, as he goes about his regular work, make the right decisions to diminish the adverse effects of the last four conditions listed above. The first five conditions, however, pertain more to the environment and tools provided to the individual programmer and can, therefore, be more readily attended to by his supervisor or the project leader.

Description and Evaluation of ADAM Production Procedures

Outside of the problem of design control, there is no more important problem than finding an adequate set of procedures for the production of a moderate or large size program. Because ADAM was primarily an experimental vehicle, little attention was paid to the development of an adequate set of production procedures. What production procedures there were had the following goals: to maintain system integrity; to make available up-to-date system components for all debugging operations; to provide a central place for card decks, tapes, and listings; to identify common problems; and to simplify system use.

System integrity was assured by administrative and programming procedures that maintained symbolic versions of the system on magnetic tape (the FORGET tape). A macro library provided a uniform definition of system symbols. A system of deck insertions based on identification cards for each routine allowed many programmers to debug with the same basic system. Problems involving system use were fielded by a "duty man" and a notebook was kept of system difficulties.

The ADAM production procedures would have been more interesting had it been possible to have the assembler as part of the ADAM System, and had the system itself been used to maintain its symbolic and object versions.

3.0 System Data Structures

Description

The set of ADAM data structures includes files, rolls, streams, and areas. Files and rolls are oriented toward the ADAM user, and are used to store the user's data base. Streams and areas, on the other hand, are intended for use by the ADAM System's programmer.

From the user's viewpoint, a file can best be thought of as a collection of information about a set of objects which have in common a set of properties. A file is thus a collection of sets of property values. Each property value may be:

1. floating point,
2. compressed floating point (not implemented),
3. integer,
4. Logical (or roll valued),
5. Query Valued (the value of such a property is computed by retrieving and honoring an associated query) (not implemented),
6. Raw Data (an arbitrary string of bits),
7. Repeating Group,
8. Subfile (not implemented).

ADAM files are serial files in the sense that file objects must be retrieved as units and appear to occupy contiguous storage. The term "file," however, is used ambiguously. The above explanation describing the user's viewpoint is the loose or general meaning. In reality, the user's file is composed of three data structures: files, rolls, and streams. Technically, the name "file" is for a specific ADAM data structure distinct from rolls and containing all types of properties with one slight exception. The value of a Logical property is stored elsewhere, and a shorthand, fixed length, internal name for this user's value is stored in the file.

A role, together with a set of "roll routines," forms a general purpose encoding mechanism which has many uses throughout the ADAM System. In general, a roll is a structure which associates names with values. One of these values, the principal value, or PV, is a shorthand name for internal use. The other values are called subsidiary values. The correspondence between names and PV's is not, in general, one to one. The totality of roll data associated with a PV is called an element. The basic concept allows the structure of a roll to be "anything at all" as long as it communicates with the user via a standard interface, but only one class of rolls was actually implemented.

Associated with each file structure are at least two rolls: an object roll and a property roll. The object roll associates object names with an internal name and the location of the object in the file. The property roll associates property names with an internal name and descriptive information which includes type, length, relative location, and so forth. Values of Logical properties which are associated with an object are stored in a Logical roll. This roll must be named by the user and he has the freedom to use as many Logical rolls as he wishes.

A stream is a set of not more than 2^{18} machine words which are identified by sequential stream addresses. When a stream is being used, some of the stream is in core and the rest is in secondary storage. The part in core exists in blocks of 512 words of 64 bits each. Stream control functions are used to locate stream addresses in core, but the responsibility of computing core addresses for referencing and changing stream data rests with the system programmer wanting to use the system.

An area is a set of machine words which exists only in core in blocks of 512 words each. Area control functions are used to allocate, expand, and release core space, but as with streams the responsibility of computing core addresses for referencing and changing area data rests with the user.

The last two types of file properties that are listed above, viz., Repeating Group and Subfile are special ones because they describe data structures. The original intent was to have the Subfile property represent complex hierarchically structured data with many levels in the hierarchy. The Subfile would be itself a genuine file and would, of course, be manipulated by file handling routines. Thus, files may be nested within files to any level. Each instance of the property Subfile relates a secondary file as a part of the primary file with the secondary file being the lower level in the hierarchy. The intent of the Repeating Group was to provide a structure for representing simple multi-valued data such as the time history of a flight plan. It was anticipated that this simple structure could be processed with simple routines producing greater operational efficiencies than would be anticipated with the Subfile structure processing. Notwithstanding the original intent of simplicity, a generalization was introduced into the Repeating Group design, i.e., Repeating Groups may be nested within a Repeating Group. With this capability added to the data structures, both simple and complex data could be represented in ADAM user files by either property Subfile or Repeating Group. The available resources would not allow implementation of both property types and so the one that seemed to require more resources to implement, Subfile, was dropped.

Evaluation

In evaluating the ADAM data structures, four very significant concepts became apparent, and although only one concept is original, they represent considerable innovation over commonly accepted state-of-the-art techniques when provided together in a single system.

The first of these is the system-wide use of general-purpose encoding mechanisms. Anyone who has built a compiler knows that the first thing that must be done to a piece of input text is its conversion from a sequence of awkward variable length strings into a sequence of objects which are easy to handle inside the machine. It is well known that for speed of compilation this is the most important design point of compiler construction, for if the passing and encoding of identifiers is done efficiently, nothing short of sabotage will slow down the compilation process. Of course, in a compiler it is necessary not only to map names into internal representations, but to be able to retrieve the original name, and to associate other values with an identifier, such as type and assigned storage location. This information is usually kept in a symbol table, and a set of general purpose subroutines exists for performing the basic operations.

There are many other applications for encoding and symbol tables besides compilers. A roll is a generalization of a symbol table, which exists not as part of a compiler or some other program, but as a separate general purpose entity available for use by a whole class of users. In ADAM, rolls are used by the translator, the DAMSEL compiler, file processing routines, the output formatting programs, and so forth. They are used for encoding property names, object names, routine names, file names, roll names, and many others. The recognition of the general utility of a mechanism exemplified by rolls is an important original ADAM idea that should not be forgotten.

The second significant data structuring concept in ADAM is the separation of data description from procedure description. It is possible to think of a computation as involving three things: the data on which the computation is to be performed, a description of the data, and a description of the computation. It is common practice in computing to combine the description of the data with the description of the computation.* Thus on a typical digital computer, to add two integers together an integer addition operation is used, whereas to add two floating point numbers together, a floating point addition operation is used. An alternative design would be to store the data description with the data, so that a single operation, add,

*Systems which allow "compools" are an exception.

would be used to add any two numbers together. The practice of keeping procedure description and data description together has been passed upward into higher-level languages. A given FORTRAN subroutine, for example, contains a complete description of all the data on which it is to operate. As far as other subroutines are concerned, this data is non-existent unless they too contain a complete description of the data, and are told in some ad hoc way that the two data descriptions refer to the same data. Within the ADAM System, an attempt is made to separate data description from procedure description at all levels where it is possible. The description of a file, for example, is not associated with any particular procedure. Associated with each file are two rolls which describe the data within the file. Queries and procedures may manipulate files without having the query or procedure contain a complete description of the data. This is an important area for consideration by designers of future programming languages, environments, and systems.

A third significant ADAM concept concerning data structures is the use of the same data structures and their associated mechanisms by casual ADAM users, sophisticated ADAM users, and ADAM System builders. Within ADAM, routines are stored in a routine file, language descriptions are stored in a language file, and a list of files is kept in a roll, as is a list of rolls. These are just a few of the ways in which data structures are shared by users and systems programmers, and represent the beginning of the recognition that system builders are users too, and have similar needs.*

Finally, it is important to note that the construction and use of the ADAM System was eased through the use of data structures of virtually infinite length. Complication was avoided by restricting to the basic routines which manipulate the data structures most of the concerns involved in the use of disk as an extension of core memory.

Of course, many problems were encountered and mistakes made during the design and implementation of files, rolls, streams, and areas. For the benefit of future system designers, the most important of these are listed below.

1. The interface between the routines that manipulate rolls and the routine users is far too complex. There is nothing

---

*Users are system builders too! A general purpose data management system can perform no more valuable function than providing the tools to enable a user to extend the system, and build their own application system.

intrinsic to the roll concept that does not allow a far simpler design of the interface. The complexity of the use of rolls in ADAM is due simply to poor design.

2. The routines that manipulate streams and areas do not assume enough responsibility for the bookkeeping of storage addresses. This is due in part to the fact that streams and areas store unstructured data which can only be manipulated by the using routine with the actual physical address. It is recommended that access to data stored in stream and area type data structures be made only with logical address through operative routines.

3. The interface between the users and both the routines that manipulate streams and the routines that manipulate areas is inconsistent and too complex. An area, in fact, could have been made a special case of a stream. There does not seem to be justification for the existence of two different types of stream pointers.

4. In any software system, and particularly in a "generalized" one, the designers and implementors continually establish trade-offs between "efficiency" (in the performance sense), and "power" or generality. In ADAM, the decision to restrict the size of a file object such that each object must fit in core, places a restriction on the number of Repeating Groups for an object. However, the 7030 has a large core and this restriction was seldom evident. Designers of future systems should cautiously evaluate decisions of this type.

5. The data structures provided for the systems programmer are not varied enough. There is a definite need for structures smaller than the block of 512 words, and a need for other structures. The block size might have been parameterized.

6. Although the file structure was able to handle a large variety of data organizations, and although the basic operations existed to process files in an arbitrary way, the capability to do this was not passed up to the experimental higher-level query languages. It is unfortunate that the experimental file generation and query languages impose a rigidity on the processing of files that is not really required by the primitive file processing level within ADAM, or that a higher level language for flexible file processing was neither fully developed nor available on-line.

Summary

Imbedded within the ADAM structures of files, rolls, streams, and areas are these four significant concepts:

1. the system wide use of general purpose encoding mechanisms;

2. the separation of data description from procedure description;

3. the use of the same data structures and their associated mechanisms by casual ADAM users, sophisticated ADAM users, and ADAM System builders; and

4. the use of data structures of virtually infinite length.

It is our judgment that most of the problems experienced in the design and implementation of the ADAM data structures are not intrinsic to the basic concepts. They are due to both detailed design and design control.

4.0 System Languages

Initially, the requirements of file generation, computation, and output formatting appeared sufficiently different to necessitate separate languages within ADAM.

In the case of simple languages within ADAM, interfaces were generally excellent. But for the more complicated functions, the language involvements brought on corresponding complexities in the interfaces between the user, the system and other languages. The level of implementation of the individual languages did vary somewhat throughout the system.

All but three of the twelve languages used within ADAM were provided to give the user ability to handle the various functions common to the design of processing systems for the management of large data bases. The functions of initial file generation, file updating and querying, and executive control of the system are handled respectively by the IFGL, FABLE, and the Utility languages. The Display language allows a significant transfer of the capability of the on-line console languages to the cathode-ray tube, light pen, and push-buttons. String Substitution and Output Formatting languages give extra ability to the user for input and output messages. LAP is the language for specifying languages for ADAM file generation and querying: the first applications of LAP defined the experimental IFGL and three versions of FABLE languages. COMFORT and DAMSEL

provide the user with the computational capability of FORTRAN and a procedure language sensitive to the user's data base, respectively.

Separated from the above special-purpose user languages are SINTAB, an entirely internal language unavailable to the user, and SMAC and STRAP, the two general purpose IBM-provided assembly level languages.

The purposes of each of the languages with comments are discussed in the following paragraphs.

IFGL

The experimental IFGL (Initial File Generation Language) creates ADAM files from an external data base stored on punched cards or magnetic tape. A set of IFGL statements define an ADAM file, the format of an external data base, and the mapping from the external data base to the ADAM file. IFGL is a simple near-English or English-like language.

Although somewhat limited and verbose, the IFGL language is easy to learn and use. It is simple to modify IFGL statements. A user need only be concerned with his own logical file structure and is spared the concern over "bit details," such as the memory allocation of stored data.

The language would have been significantly enhanced if some features for conditional control were available so that it could perform more of the functions that for ADAM were done by FORTRAN pre-processing. Also, one should have included more elaborate diagnostics and error-handling features.

FABLE

The FABLE (First ADAM Basic LanguagE) language was originally intended as an experimental interim, simple near-English language for use by an applications-oriented person at an on-line console for file querying and file updating.

FABLE is more advanced and powerful than most query languages. Its utility was significantly increased when additional desirable features, such as cross-file referencing, were incorporated into the language in later versions. The simple features of the basic FABLE language are easy to understand and use. It becomes more complex and more difficult to use as more of its sophisticated features are employed. This may be due in part to the lack of standardization in construction of statements and in use of punctuation and key words,

or perhaps because the FABLE approach is not the best way to express complex procedures.

FABLE is a procedural language, but must be considered deficient as a programming language. Several users with previous programming experience have commented that restrictions were sometimes felt due to the lack of some form of conditional control capability.

Experience has shown that user personnel with programming backgrounds find that a higher order, near-English computer language, such as FABLE, is relatively easy to use for most tasks, provided the constraints are well documented. Formal training facilities and documentation were not provided Non-programmer users are not accustomed to expressing their ideas following rigid rules; therefore, they may have trouble in learning to write a long sequence of steps for a comprehensive file query without some clerical, logical, or functional error.

Knowing some of the algorithms by which files are manipulated gives even the user with little or no programming experience much more control in using the language for his purpose.

$UTILITY

The Utility language allows the user to do simple executive tasks requiring minimum ADAM System involvement. Examples of such tasks are time logging, tape mounting messages, job and task termination, and activation of numerous debugging aids in the system. Correspondingly, the language is extremely easy to use and because of the philosophy of system avoidance in its implementation, rapid responses are realized.

DISPLAY

The Display language provides the ability to use the CRT with its associated light pen, and the push buttons in an interactive manner as an alternate input device for stored source queries. These queries are stored in a file called the CEMETERY file. By means of this language and its implementation, a user operating at a display console is provided with string substitution and parameter insertions to the FABLE and IFGL statements. Selections of query language statements, including substitutions and insertions, are accomplished via light pen action in an extremely rapid and flexible manner.

Perhaps, more accurately, this should not be called a language, but a Display Input Capability which is supported by many languages and facilities of the ADAM System, that is:

1. the Output Formatting Language, to describe how an output may be used as input;

2. IFGL, to generate the CEMETERY file;

3. the Rule Recognizer, to route display inputs;

4. FABLE, into which display inputs are eventually converted;

5. Skeleton language, in which the skeletons stored in the CEMETERY file are written.

If there is a language to be described in the display input capability, it is the Skeleton language, last mentioned. However, it should be noted that the interesting aspect of the display capability consists in seeing how all these pieces work together. The philosophy under which the rest of ADAM was implemented is that ADAM is non-interactive, yet by utilizing its mechanisms astutely, the interactive non-typewriter input capability was modeled.

String Substitution

Use of the String Substitution language simplifies input messages sent to the ADAM translator by allowing the user to define an equivalence between a word and a string of words. This string of words can be a partial or full query with parameters inserted. The language is easy to understand and was used extensively.

Output Formatting

The Output Formatting language allows the ADAM user to define a procedure for transforming data in a file into a suitable form for output. The notation for specifying formats is that of a macro language consisting of a set of operators and associated parameters allowing one to manipulate file values and literal information into a desired representation on output devices.

Although the output formatting language is easy to use, the language statements do not lend themselves to direct visualization of the final format of the output display. The choice of the language operators was quite complete and they facilitated manipulation of output data. However, some form of conditional expression, providing, for example, the ability to skip operators during an output process would have been valuable, according to several users. Because the only diagnostics were those of SMAC/STRAP, it was occasionally difficult to relate them to erroneous statements in the formatting language.

LAP

LAP is a language for defining user languages for file manipulation. It was intended to ease the expansion of the set of ADAM languages by providing a vehicle for the formal definition of syntax, and a set of tools for mapping syntactic elements into appeals to a set of basic code generators. LAP has been used to implement the experimental languages FABLE and IFGL. The LAP definition of the syntax of a language is assembled off-line by the LAP assembly program.

In the construction of a new language or language feature, a thorough analysis is brought about by using the simple and precise LAP diagrammatic approach. Translating the information from the diagram to LAP statements is only a clerical process. Many major changes to FABLE and IFGL were expressed with little effort in LAP.

Although it is easy to define the syntax of languages in LAP, doing so requires a knowledge of the existing generators which, unfortunately, are not well documented.

FORTRAN/COMFORT

The FORTRAN/COMFORT language is a subset of FORTRAN, with certain restrictions that enable its compiled output to be integrated into the ADAM routine file. This language requires modified FORTRAN CALL statements necessary for ADAM compatibility. With a few exceptions, the entire capability of FORTRAN is available. The most awkward feature in using FORTRAN is the passing of information to and from the rest of the ADAM system via the interface; users have commented on the excessive clerical operations required to associate properly FORTRAN data with ADAM data. Only separate undimensioned variables can be passed through the interface. The data structures that were included for convenience of storing and processing, i.e., files and rolls for ADAM, and arrays for FORTRAN, cannot be passed back and forth. FORTRAN routines which had been integrated into the ADAM routine file were, nevertheless, not treated in the same way as other ADAM routines. When appealing to an ADAM routine, it is not necessary to specify all the routines that may be appealed to by the one that is first requested, however, when appealing to a FORTRAN routine, a complete list of routines must be provided to the monitor that includes not only the one being called, but all those that it may call.

The inclusion of FORTRAN capability for ADAM was a retrofit and this is reflected in many of the inconveniences encountered in its use. Including this capability in the initial design consideration could have eliminated many of the difficulties.

Users commented that they felt that the interface difficulties were only inconveniences, and having the computational capability of FORTRAN was essential.

DAMSEL

DAMSEL was originally intended to be a language for writing system routines and specialized user procedures to be entered into the system. The statements were designed to provide convenient ways of specifying the common operations associated with the ADAM System and can explicitly reference the names in the user's data base. In addition, DAMSEL statements may be intermixed with SMAC and STRAP statements.

There was implemented only that subset of the DAMSEL language which allowed the user to do arithmetic and a restricted set of file manipulative operations. The SMAC/STRAP intermixing feature eased significantly the difficulties of a partially implemented DAMSEL and some of the interface difficulties with FORTRAN. Due to limitations in the SMAC and STRAP processors, DAMSEL was an off-line compiler and required a separate job outside of ADAM.

Originally it was intended that the DAMSEL compiler be part of the main ADAM System and thus be available to any routine requiring translation. This approach was far too ambitious for the available resources, and the present version utilizing SMAC and STRAP was begun. In view of the limited implementation, the limited use, and the inconvenience of separate jobs, it is not clear that this approach enhanced significantly the research or application of general purpose data management systems.

SINTAB

SINTAB is a procedure language consisting of operation codes and parameter references and is internal to the ADAM System. The code appeals to system routines to allocate storage, handle files, rolls, and formats. This code is the object code of FABLE and IFGL messages and becomes the source code for the interpreter which performs the processing required by the input message in IFGL or FABLE. Because it is generated dynamically during the execution of a task defined by an input message, the user is unaware of its existence.

The SINTAB language format is highly packed and rather non-uniform. It contains a general conditional operation and a general control transfer operation, as well as the facility to provide variables, functional arguments, and access to any routine in the ADAM routine file via a generalized routine call mechanism.

The implementation of SINTAB would appear weak as the object language for arbitrary languages defined for processing by the ADAM translator, in that it provides only a rudimentary procedure mechanism to support hierarchies of procedures defined in a higher-level language. It also appears weak in providing scope or block structures for delimiting the range of context of symbols.

SINTAB routines may have effectively infinite length, since there is a paging mechanism (provided by software) superimposed on them. This paging facility represents a desirable feature easily available in an integrated, generalized system because this basic functional capability is economically provided by system-wide service routines.

SMAC/STRAP

The 7030 SMAC macro language, written by IBM and modified by MITRE, facilitates the writing of problem programs. The macros are translated into STRAP symbolic statements. The STRAP statements, whether generated manually or by SMAC, are then translated into relocatable or absolute 7030 machine code.

SMAC includes a library of IBM-supplied macros as well as language statements for generating new macros. For ease in coding, the processors have been designed to accept intermixed SMAC and STRAP statements.

The majority of code for building ADAM System routines was written in these languages. The MITRE modifications to SMAC, although unable to do much in improving the communication between instances of macros, were able to increase significantly the general freedom within the macro generators themselves.

Summary

It was thought at the outset of the ADAM project that the requirements of file generation, computation, retrieval, and output formatting were sufficiently different to necessitate special problem-oriented languages and possibly separate translators. It is our opinion, in retrospect, that this is incorrect.

Our experience leads us to believe that the requirements of file generation, computation, retrieval and output formatting can, and should be, blended in a single language.

Our experience with the implementation of languages showed that each of the following design decisions must be considered separately and should be decided independently of the others:

1. whether the language is to be employed by the user on-line via a query station, or off-line by the user via card input;

2. whether the language is to be compiled or interpreted;

3. whether the language is to be procedural or non-procedural;

4. whether a great deal of attention is to be paid to the efficiency of execution.

The independence of these factors should not be underestimated by designers of future systems when providing for the user capabiliites for trade-offs and optimization.

5.0 System Language Manipulators

In order to implement the languages of the system, the following manipulators were employed.

LANGUAGE	MANIPULATOR
IFGL	ADAM Translator
FABLE	ADAM Translator
Utility	Rule Recognizer, Rule Routine
DISPLAY	Non-Typewriter Input Routines
String Substitution	String Substitution Mechansims
Output Formatting	SMAC Processor, OUTFOP
LAP	LAP Assembler
FORTRAN/COMFORT	FORTRAN Compiler and COMFORT Post Processor
DAMSEL	Compiler
SINTAB	Processor
SMAC	STRETCH Macro Processor
STRAP	STRETCH Assembly Program

TABLE A

In considering the language manipulators, one may ask the question: Is the choice of this set of language manipulators a good way to provide the language translation capability desired? The answer is yes, but this must be qualified. With the exception of the SMAC processor, the

STRAP assembler,* and the partially implemented DAMSEL compiler, the language manipulators provide the specified language functions and do so with reasonable performance and moderate freedom from error.

The manipulators were designed and developed by different people and to meet the separate language requirements at different times throughout the life of the project. Some of the manipulators are excellent, while others have features that strongly detract from their value. In the implementation of the ADAM translator, there were many mechanisms, new data structures, etc., which unfortunately were not incorporated into the system at large.

A variety of approaches was taken in the language implementation. In most cases, the choice made to build a compiler or interpreter was a good decision, even though the particular embodiment may have included a few undesirable characteristics.

In an evaluation of a complex system such as ADAM, judging each part by itself and also the collection of parts as they exist is reasonable, but also significant to the research nature of the project is the experimentation and attendant learning which may not be readily deduced from such evaluation results. From today's vantage point, an operating ADAM System and with the experience thus derived, a key question to be answered is: What would be changed if the ADAM languages and their manipulators were to be done over?

The number of languages would be reduced sharply, to about three or fewer, with a corresponding reduction in language manipulators. These languages would include all the present language capabilities integrated within them.

From our experience with applications, it became evident that two levels of performance are desirable: the first while experimenting with queries, procedures, and data structures to design the procedure; and the second while testing the procedure over many objects in the files. The manipulation capability would best serve the user and builder if it allowed for options to select flexibility or speed, and the saving of source and object code.

By flexibility, we mean that the language capability has a richness in the expression of a wide variety of functions and mentioning of objects. When a flexible language is to be implemented, we think

---

*SMAC and STRAP are not being evaluated in this document.

it is reasonable that the implementation should save a maximum amount of information in the system so that the system can communicate with the user in source language. By speed, we imply the most efficient object program which the system can generate. In either case, it is desirable to be able to save object code to avoid retranslation, and source code to avoid reentering the entire query or procedure.

The most important conclusion to be derived from our ADAM language experience is that the system must maintain itself and be written in one of its own languages. Our experience allows us to conclude that because the environment and circumstances are so complex for the coding of any function, the language level of the coding should be high enough to help reduce errors. In this vein, the DAMSEL language was designed to meet the needs of the system programmer and allow convenient expression of his algorithms and data. Thus, not only could the user employ it for specialized routines, but an ADAM System programmer could design the entire ADAM System, use DAMSEL to write the system, and have it compile itself.

The enormity of the task of implementing the full language was too much for the resources available and only part of the language was attempted. The compiler for even the subset is of such generalized capability that only near the end of the ADAM project was it checked out well enough to have the execution of its output code approach any degree of reliability. Thus, unfortunately, the ADAM System was not coded in DAMSEL.

6.0 Memory Resource Allocators

Memory Allocations

With few exceptions, memory assignments are made by two allocators: the secondary storage (disk memory) allocator, SESCON, and the primary storage (core memory) allocator, MARASS. Two functional routines, BASAL and CLOD, transfer between primary and secondary storages data and routines, by appealing to these primitive allocators.

Disk Allocations

The primary purpose of the ADAM routine in charge of disk allocations is to make reserved and semi-reserved allocations of physical blocks on the disk for continuous logical storage. Each new allocation is given an identification, thereby allowing the user to access disk without knowledge of the current physical assignments.

In addition to new allocations, the routine will delete or increase existing allocations and will read from or write to disk storage. User's

references are logical and relative to the start of logical storage, not physical disk assignment.

Although an allocation may consist of a sequence of non-consecutive disk blocks, the allocator performs any necessary linking, so that the storage space appears to be uninterrupted. The user, then, need only concern himself with the identification of the disk allocation and its size. This interface proved simple to use and made the user relatively independent of the hardware and details of housekeeping.

For housekeeping purposes, disk locations can be reassigned and discontinuities removed or at least reduced by the temporary buffering of data on magnetic tape storage.

Core Memory Allocation

There are two basic types of core allocation, one for routines and one for data, which employ different methods.

Routines, once allocated in core memory, cannot be moved, whereas data can be moved by the allocator. Routines are allocated from low toward high-numbered memory locations in 64-word increments and data are allocated from high toward low-numbered memory locations in 512-word increments called pages. This allows the amount of space occupied by each type of core allocation to be dynamically varied. Operation has shown this flexibility to be very useful.

Routine Allocation

When a routine is designated as fixed or allocatable, the meaning is that it is resident in core throughout the operation of the system or is resident on the disk and transferred to core when needed.

Fixed Routines

Fixed routines remain in core during system operation and reside in the lowest numbered core locations. The list, in loading order, of fixed routines is stored in a table used at system initialization time. The list was chosen to include the most frequently used routines so they would not have to be dynamically allocated and reloaded, and also those routines which were multiprogrammed and of necessity had to remain in core.

Later in the project, the Loading Order Table for fixed routines was supplemented with another table that allowed the dynamic release and reassignment of fixed routines. This feature was implemented specifically for the AFLC project and was found invaluable.

Allocatable Routines

When an executing routine requests that another routine be loaded, a system program, CLOD, allocates core memory, transfers the proper blocks from disk, and performs the necessary modification of the transferred code by calling the appropriate system routines. When an allocatable routine is no longer needed it can be dismissed.

Data Allocation

Although data allocation in core memory is performed by BASAL, it in turn is controlled and supervised by those system routines that manipulate the data structures. For example, system routines automatically attended to keeping file data and file structures, which were both stored somewhat jointly in associated ADAM data structures, called Files and Rolls, properly meshed for several files simultaneously.

Evaluation of Allocators

The generalized allocation schemes proved extremely valuable to the ADAM development by relieving the system programmer (and the user with specialized procedure requirements) of the tedious tasks of housekeeping both data and routine allocations in primary and secondary storage. Similarly, he was spared the housekeeping chores for manipulation of data structures.

Although there were many excellent features of the allocation concept, the paging mechanism was, unfortunately, sensitive to the type of information being transferred, viz., routines or data, and not sensitive enough to the availability of core memory. The core allocation algorithm does not utilize those available 64-word increments that lie between other increments assigned to active routines.

One may consider the internal housekeeping functions of ADAM to be data management functions, where the data (and resources) managed represent the programs, internal data, and so forth, of the system itself. Considered in this light, and in view of the promise of generalized data management techniques in the processing of higher-level data, it seems appropriate to recommend that designers of future systems seriously evaluate and further explore the possibiliites of reflexively applying generalized techniques to the management of the system itself.

Fortunately, the designers of the ADAM System considered that storage allocation was an essential concept for the system. Its implementation turned out to be an effective system mechanism in

most cases, especially when used by other parts of the system such as the routines for manipulating data structures. This implementation, however, included one feature that quite often caused difficulty. Optional control for dismissing routines that were no longer needed in core was given to the programmer. When he forgot to include coding for the dismissal of the given routine, the end result was that soon all of core memory was indicated to the allocation routine as being unavailable.

Our experience with the operation of the ADAM System indicated one difficulty in the design of the allocation scheme which would prevent the system from operating for any extended period of time with a reasonable activity level for file manipulation and creation. In the allocator there is a table of fixed length in which is kept pertinent information concerning the disk allocations. Whenever logically related data, such as in a file or roll, are stored on the disk in consecutive blocks, a certain amount of data is entered into the table. More data is required whenever the disk blocks are no longer contiguous, and the disk storage of the file or roll is then said to be discontinuous. There are no automatic housekeeping routines available for reducing the discontinuities, and with the fixed length table, troubles were encountered on some of the long runs of the system in which there was a fairly active generation of storage discontinuities.

7.0 Summary

The design decisions that are given below are only some of the numerous decisions that were important in the design and development of the ADAM System and they are given in no special sequence.

It was decided that the ADAM data base would hold in common the data and routines (in object code) for both the system and the user. Dynamic storage allocation was decided upon and would include the allocation of core and disk memory. Additionally, it was decided to have data relocatable in core memory, and routines not relocatable, on a single allocation. The IBM SMAC compiler and STRAP assembler programs were the software tools to be used for coding the ADAM System. Closely related was the decision to design and construct an ADAM compiler sensitive to the needs of the programmer building a system such as ADAM. There was a decision to have the on-line languages translated by a syntax-directed compiler. It was decided that there would be four data structures within ADAM; files, rolls, streams, and areas; and early in the project a design of file and roll structures was set down. Nearly a year after the start of the ADAM project, there was a decision to have the output of a query

structured in the format of a file so that the system would be able to maintain information from one query to another.

The formats and media for data input during file generation and output for report generation were to be specifiable by the user.

Seven of the sample of nine design decisions given above are good to excellent, for they bring into the design many very desirable concepts. Embodiment of these concepts provided capabilities to the user to not only do many of his jobs with extra facility, but also to adapt the system to meet special needs, such as modifying the on-line languages or writing special computational routines.

Certain aspects of the ADAM conventions for routines seem subject to debate. Within different environmental conditions these conventions might obviously be inappropriate. In particular, the use of pure coding techniques, non-absolute procedure calls and returns, relocatable (including dynamically) procedures, facility in the invocation of recursive and co-processor routines, system-managed variables, arrays, and other data structures should be seriously considered in any future effort of this type.

In a general context, the ADAM design included provision for four basic capabilities that should be made available to a user by a general purpose data management system:

1. definition of the user's own query language,

2. acceptance of input data for file generation in a variety of formats and media,

3. specification of output formats and media for report generation,

4. acceptance of routines written by the user for his special needs as part of the system.

The two decisions to relocate data, but not routines, and the adoption of SMAC and STRAP, do not aid as do the other seven decisions in reducing unnecessary complexities and restrictions, and thus are antisystemic. Distinguishing between data and routines at the basic conceptual level and so early in design, allowed two incompatible allocation algorithms to develop, as well as two file handling mechanisms. This unnecessary duplication was also accompanied by another condition that slowed down the checkout of a program. Because routines would never be relocated in core once allocated, many of them became self-modifying. When a routine can modify itself, it is more difficult

to debug because of the volatility and scattering of the evidence for debugging. A further consequence, but of less significance, is that with self-modifications occurring here and there in service routines, the possibility of writing recursive routines is made more difficult.

STRAP, and its pre-processor SMAC, were written by the manufacturer with the philosophy that the entire resources of the machine were available and under their control while operating. This extreme arrogance did not allow them to become part of the ADAM System in such a way that a part of the ADAM System could appeal to them for translation while ADAM was in operation.

The design of too few of the parts of the ADAM System was documented well and in an orderly fashion: first appearing as a proposal that was discussed and agreed upon, then revised and issued as a specification. Other parts, however, were implemented directly from unrecorded considerations of the required functions; and thus design and implementation often became one: prevalently, an interface was specified by the first person who needed it. Some significant benefits may have been derived from having at a very early stage a concise document containing:

1. general system specifications,
2. an enumeration of the subsystems with the major responsibilities of each,
3. rules and conventions for interfaces,
4. conventions for error handling,
5. instrumentation requirements for routines, and
6. guidelines for designers of subsystems.

Although much of this material did appear later in the project, it was written in considerable detail and was more concerned with the separate subsystems rather than the system as a whole.

D. SUMMARY OF PROBLEMS NOT INVESTIGATED

Though the ADAM experiment was ambitious, there are many facets of data management which were outside the scope of its activities and thus certain aspects of the real-world were not addressed in the realization of ADAM. The following comments, then, are concerned with these aspects. They are presented as examples of critical subjects that are not clearly highlighted by ADAM experience, but are well known in the technical community. Consequently, conclusions drawn from the ADAM experience to be discussed later must be interpreted with these thoughts in mind.

Quality Control of Data

A data management system first needs data to manage. Contemporary data management systems require large amounts of data. The acquisition, correction, and verification of a large amount of data is a first-order problem, certainly equivalent in magnitude to the development of a large set of procedures comprising a complex program. ADAM, even in its application experiments, never encountered data that is wholly representative of the real world. Real-world data is not nicely structured. Data items, for example, may have errors in value, be erroneously omitted, or occur out of sequence. In jargon, data is often called "grubby," for such reasons. When dealing with a data set of one, or ten, or one-hundred million characters, attaining adequate confidence in the integrity of the data set becomes of extreme concern. It is understandable that ADAM did not consider this problem in view of the size and requirements for resource expenditures.

Secure Data

Although ADAM was used successfully in a remote manner, one should not consider that the solution to all problems associated with such operation is in hand. At no time in these experiments was sensitive information transmitted by the system. The related cryptographic problem is recognized as real, but was not examined by ADAM.

Similar comments may be made about the problems of authorized access (for retrieval or change) to subsets of the ADAM data base or procedure base. Ambitious attacks on these problems are currently being made in the technical community, utilizing esoteric software and hardware techniques, but the implementors are not yet ready to announce success.*

---

*The MIT MULTICS system is an outstanding example.

Error Handling

ADAM was implemented in a laboratory environment. The majority of its users were relatively senior people--capable of a high degree of adaptation to the system. Perhaps for this reason, and others, the general problem of error handling received less attention than seems desirable for a production program.

Data Representation

ADAM is a data management system. One might inquire: What kind of data does ADAM manage? Casually one might reply: Numeric and alphanumeric. But such a reply would be naive. Consideration of numeric data alone quickly leads from integer and floating point types to rationals, complex numbers, variable-precision numbers and a host of others. How does one encode and manipulate time, latitude/longitude, angle measurements, etc.? Obviously numbers alone provide a very rich data area. ADAM made but a weak attempt at attacking this area, a field that attracts continuing research.

Garbage Collection

Contemporary techniques for the construction of generalized systems use methods which tend to produce "garbage." Garbage is data which is no longer being managed. As such, it is of no concern to a data management system, but the memory space (resource) it occupies is of concern when it grows large relative to the available storage resources remaining at any one time. ADAM tends to produce less garbage than many contemporary systems because there are a few automatic mechanisms in the program. These separate and specialized tools are augmented by some options available to the user in the utility language for other types of cleanup. The garbage collection problem requires more than this because any residual garbage will eventually choke a system. Unfortunately, today's technology has not yielded a satisfactory solution to the garbage collection problem.

Breakpoint

Finally, ADAM was not directed toward the "breakpoint," "roll-back," or "un-do" problem. This problem is essentially created by the necessity of being continually prepared to retreat to a previous state when data, logical, or hardware problems force the premature termination of some operation. In such a case, it is probable that inconsistencies or errors have been introduced into the data set. Often such problems may be rectified only by reverting to a previous instance of the data set. General techniques for dealing

with this problem were not attempted during the ADAM project although some commands in the utility language permitted, at the user's discretion, the copying onto magnetic tape of the present version of the system including the data base.

SECTION V

CONCLUSIONS

The conclusions are divided into two parts with those concerned with applications aspects being presented first followed by those concerned with system aspects.

SUMMARY OF APPLICATIONS ASPECTS

Although some aspects of data management were ignored in ADAM, a number of valuable conclusions can be drawn. ADAM was applied to several problems that are appreciably realistic, and will now be examined from the experience of these applications.

Value of General Purpose Data Management

The applications of ADAM give rise to both optimistic and pessimistic observations concerning the notion of a general purpose data management system. The total experience of the project does not conclusively establish or discredit the general purpose approach as an effective way to achieve a system of wide applicability. The promise held out by the ETF effort years ago still remains. However, current industry-wide interest in ADAM-like systems may lead to a confirmation or denial of the generalized data management concept as a consequence of greater experience and effort with these systems.

It has been demonstrated by ADAM that substantial economy in construction time for applications programs can be realized by the use of a general purpose data management system. Similar economy can be realized in the production of certain modeling systems, i.e., representations of proposed special purpose data management systems. In general, the trade-off being made to achieve such economy involves the cost of implementing the general purpose data management system itself as well as the performance characteristics or efficiency of the programs comprising the general purpose data management system. An ADAM-like system can usually be partially modified or "tuned" so as to improve its performance on a specific task. The basic trade-off between generality and efficiency is, however, in the hands of the ADAM System designer, and not the user. Thus the performance loss for particular applications may negate the advantages gained in lead-time economy. A need exists for a mechanism to allow more extensive and convenient user control or direction of the trade-offs involved.

The ADAM System provides an environment to support the construction of application programs characterized by a tendency to evolve. Such features as the separation of data description from procedure description, maintenance of files by the system, and automatic maintenance of resources of the system are conducive to the evolutionary development of programs. Inasmuch as so many application programs have this nature, an advantage accrues beyond the lead-time gain in initial construction.

Data Base

The ADAM data structures, particularly the file structure, have demonstrated a true usefulness for the organization of realistic data. By direct application it was shown in the ADAM experiment that the file structure accepts a wide assortment of data with relative ease. This is not to say, of course, that the ADAM file structure is necessarily adequate for the convenient, efficient handling of an arbitrary data base.

There were times when the data structures provided by ADAM were not appropriate to the task. In particular, the absence of arrays in ADAM was quite awkward on occasion.

ADAM is essentially a disk-oriented system. Such a system tends to support random access to data and procedures. But the restriction of data base size implied by a disk-oriented system is not necessarily insignificant. Very few realistic problems have data sets of less than five million characters, the ADAM restriction. The classical solution to this dilemma has been the sole use of magnetic tapes for data storage. But this attack has obvious problems in access time, at least. It appears, therefore, that a hierarchy of storage devices is required, including disks and tapes at least, and possibly drums, data cells or magnetic card storage, or bulk core storage.

ETF, the predecessor of ADAM, was characterized by separate file generation and retrieval facilities implemented on different computers. The integration of these facilities in ADAM is one of its significant contributions to the general purpose data management system concept. However, this integration led toward a problem not initially foreseen in the development of ADAM. This problem is known as the massive update problem. Within the system little acknowledgement was given to this problem initially. The ability to accept large quantities of update information while the files are being maintained by the system in what is essentially a random access, highly-structured fashion is a problem that must be faced by any non-experimental general purpose data management system.

Solutions to this problem typically are easy in systems which are optimized towards file generation, and difficult in systems which are optimized towards rapid retrieval.

Input-Output

The ADAM System contains multiprogrammed input and output mechanisms by which the system was able to support more than one user on-line. In some respects, the system appeared to be time-shared. However, the execution time allotted to a user program was not constrained to a specified time quantum. Any query processed by ADAM was scheduled as one continuous task until the occurrence of an input-output operation. Interaction with a common data base by several people in a concurrent fashion was also not demonstrated by the ADAM System and extrapolation to such use cannot be done lightly. Substantial logical extensions of the system design is involved to achieve such operations.

During the AFLC experiment, users occasionally posed queries to the system which created voluminous amounts of printed output. The output processing performed by ADAM is multiprogrammed, but it was possible to select an output device for a particular report and thus tie up that device until total completion of the report production. It appears desirable consequently, to provide special abort or redirection facilities for cases where massive output may occur.

In the AFLC experiment, a remote station with a teletypewriter and a printer was the total user interface with the system. In this case, no alternative output devices were available at the user site. A simple query could then occasionally produce large quantities of output which impeded the use of the system by successive users. The remote installation during the AFLC experiment required a special adapting device at the computer and commercially available devices at the remote site. Commercial communication lines were used for data transmission. This was possible since all data processed and transmitted by the system was nonclassified. The restriction on the total assortment of output devices was limited by economics in this case. The installation and use of the system at the remote site proved quite successful.

In one form of operation, ADAM provided an on-line interface to analysts. As such, the capability for the analyst to examine the various properties of the data base was extremely useful. This is somewhat analogous to the efficiency that can be achieved by providing interactive programming support services for the development of computer programs. The ability to examine new attributes

of the data base casually, and to examine the various paths which are created by decisions during the process of examination can prove extremely useful.

The addition of display facilities to the ADAM System indicated the great power and usefulness of graphical techniques. In addition to being necessary for ADAM as a modeling tool for arbitrary command systems, the display facilities proved worthwhile during casual use of the system.

Languages

The two ADAM experimental languages devoted to file manipulation are IFGL, a file generation language, and FABLE, a retrieval or query language. Examination of the FABLE language evolution through versions I, II, and III indicates a trend toward increasing capability for cross-file referencing. The ability to reference several files concurrently was found to be required in a general purpose data management system. Indeed the total number of files that can be processed in this manner should be quite large.

Serious effort was given in IFGL and FABLE to separating data base description from programming statements. However, the use of the group, subgroup, property notation in these languages, and the positioning of statements within the languages are then directly dependent on the structure of the data. The separation of data description from program has thus not been done with 100 percent success. This problem is recognized as a difficult one, and a satisfactory solution to the problem of separation in a general purpose data management system might yield high return, especially in a large programming effort.

Both the IFGL and FABLE languages were relatively weak as algorithmic programming languages. Of course the primary intent in their design was the creation of languages to implicitly realize reasonably simple and routine manipulations for the casual user. So long as the required functions were indeed quite simple and routine, these languages were found to be satisfactory. It should be noted, however, that many mission-oriented people had difficulty in using FABLE even in applications well within its capabilities. This may be due to the early unavailability of adequate training and documentation for ADAM users.

The ADAM experience indicates a necessity for a programming language suited to the expression of complex algorithms. This need arises in the file generation, update, and retrieval situations.

The file generation capability in ADAM, represented by the IFGL language, proved inadequate for interfacing with arbitrary input data sets. As a result, pre-processors were often built for subsetting, data verification, and format changing functions.

In addition to the near-English languages IFGL and FABLE, all the other user languages except the LAP language, were used by the application programmers for their specialized needs for controlling the system, mathematical computations, formatting output, and display console activities. Their evaluation, although strongly supporting the worth of the language features, is that there were too many separate languages, and combining the features within a fewer number of languages would be more desirable.

SUMMARY OF SYSTEM ASPECTS

Introduction

Most of the activity and concern in the ADAM project was due to the fact that ADAM is a large system of programs. The following conclusions are derived from the experience of the building of the system.

Support Software and Procedures

ADAM has demonstrated that the construction of a general purpose data management system represents a large programming task and requires commensurate support software and production procedures. Unique, or essentially unique computers aggravate the amortization problems associated with the development of such large scale support software packages. Although MITRE modified the software considerably, it was still not extensive enough to support a very large programming system.

Automatic aids for programming and debugging were developed, not only early in the project activities, but as an on-going operation throughout the project. It is not possible to predict the needs for software tools to the point where all tools and procedures are identifiable and can be specified at the start of a research project, but some manpower must be allocated to meet these programming needs that develop during the project. The ADAM project demonstrates that significant benefits can be gained by automating major parts of the control of the symbology of the system itself. Specifically, in a case like ADAM, which is a system for general purpose data management, the system should be self-managing of all its symbolic form rather than the limited amount managed in ADAM.

Application Area

The target application area should be carefully defined in order to provide the basis for adequate specification prior to the design effort for the programming project. Care must be exercised, however, in deriving realistic and obtainable requirements. Once defined and built, the system should be steered away from applications not within its context.

Programming Arrogance

General purpose data management systems have demonstrated their utility in certain problem areas. The use of these facilities as an integrated subsystem in a larger system should not be excluded by the adoption of a philosophy of arrogance in the design of the general purpose data management system, its components, or its supporting system.

Internal Communication

Key importance in a large programming project must be placed on the attempt to maintain simple and uniform interfaces for control and data between component programs. The type and size of data forms, data form manipulators, and the kinds of conditional program control features that are used by a programmer in one area of the system may be convenient for programmers writing other parts of the system and thus should be considered as candidates for system-wide use. Further, especially in the case of a general purpose data management system, many of the needs of the user and system programmers can be served to a greater extent by making available to both of them all the data forms, form manipulators, programming features and languages that are often considered the private property of one or the other.

Languages

The experience of working with the many languages in the ADAM System justifies the conclusion that although the number of languages and language manipulators should be reduced, the features contained within these languages should not be reduced.

The ADAM translator that accepts IFGL and FABLE input is a very powerful translator inasmuch as it can accept definitions of many languages via their descriptions in LAP notation. The design of the part of the translator that generates the object code, however, attempted to accomplish too much. This resulted in generators that were too large and too detailed to be easily used.

More properly, the generation of object code should be divided into two separate functions: generation of symbolic code and final assignment, such as an assembly program. With such a change, the ADAM translator would then be able to handle most, if not all of the translation requirements for the entire system.

Input-Output

Conclusions on input-output may be divided into three parts: intersystem compatibility, query language capability, and internal programming for I/O. Although ADAM was intended to be able to receive data from and generate data for other systems, it cannot be said that success or failure has been demonstrated conclusively. ADAM was used as a subsystem for the Tactical Airborne Beacon System and communicated in real time with other major subsystems.

Applications programmers selected, for economic considerations, the approach of pre-processing the FORTRAN programs, the large and complex data base from the AFLC operational system, rather than exhausting all possibilities of performing the subsetting exclusively with the ADAM System.

Special input-output conversion routines and the ability to specify output formats significantly aided in the success of both IFGL and FABLE experimental languages.

The philosophy adopted in the throughput part of ADAM is that the input-output message itself is the important thing and should remain separate from format and device information as long as possible, that is, as soon after input and as late before output as possible. This philosophy is reflected in the ADAM programs and thus allows the dynamic selection of input-output devices, as well as addition of new devices to be a simple operation.

The ADAM Concept of the Repeating Group

A very substantial part of the ADAM effort was devoted to the design and implementation of the repeating group. It has great significance in applications because of its ability to accept as properties any property types including nesting Repeating Groups and because of its ability to accept any number of repetitions limited only by core memory size. The systems aspects of this unique subfile structure are also significant inasmuch as it was the single concept which consumed more of the ADAM project resources to develop than any other. In the opinion of some, it is the most distinguishing feature of the ADAM approach. There are some disadvantages because of the fact that it requires special routines

for Repeating Group handling in addition to the routines for file handling. This apparent duplication of effort was not undertaken lightly, in view of the technical problems in the design and implementation of such capabilities. The Repeating Group was originally intended for the representation of simple multi-valued data which would not itself have Repeating Groups as properties, i.e., non-hierarchical data. The property type "Subfile," which was a genuine file, was intended to contain the hierarchically structured data. Because the available resources would not permit the implementation of both property types, the type Subfile was dropped because it was more difficult to implement. Although opinions are sharply divided on the question: Was it right to adopt the Repeating Group in preference to Subfile as a file property?, there is unanimous agreement that a general purpose data management system should include at least one of them.

Documentation

The necessity for a good documentation scheme was recognized early in the ADAM development effort. The general documentation problems for systems such as ADAM are not solved.

APPENDIX I

BIBLIOGRAPHY

1. ADAM Project, "A User's Guide to the ADAM System," ESD-TR-66-644, The MITRE Corporation, Bedford, Mass., August 8, 1966.

2. Baum, C. and L. Gorsch, eds., Proceedings of the Second Symposium on Computer Centered Data Base Systems, SDC TM-2624/100/00, System Development Corporation, Santa Monica, California, December 1, 1965, pp. 3-87-3-121.

3. Beilin, I., "MERGE," ESD-TR-67-371, The MITRE Corporation, Bedford, Mass., August 1967.

4. Buckles, G. A., Lt., USAF, and S. B. Carpenter, Lt. USAF, "Computer Centered Data Base Systems," ESD-TR-66-499, Directorate of Computers, Deputy for Engineering and Technology Electronic Systems Division, Air Force Systems Command, United States Air Force, L. G. Hanscom Field, Bedford, Mass., October 1966.

5. Char, B. F., and Capt. A. Foreman, ESRC, "Final Report--Joint AFLC/ESD/MITRE Advanced Data Management (ADAM) Experiment," ESD-TR-66-330, The MITRE Corporation, Bedford, Mass., September 6, 1966.

6. Clapp, J. A., "A Description of the Internal Operation of the ADAM System," ESD-TR-67-372, The MITRE Corporation, Bedford, Mass., August 26, 1967.

7. Connors, T. L., "ADAM--A Generalized Data Management System," Proceedings, AFIPS Spring Joint Computer Conference, 1966, Spartan Book Co., Washington, D. C.

8. Connors, T. L., "An ADAM Presentation," MTP-16, The MITRE Corporation, Bedford, Mass., October 1965.

9. Connors, T. L., "Software Concerns in Advanced Information Systems," Proceedings of the Third Congress of Information System Science and Technology, Thompson Book Company, Washington, D. C.

10. Foreman, A. C., "Advanced Data Management Experiment," IEEE Transactions in Aerospace and Electronic Systems, AES-2, January 1966, pp. 115-120.

Unclassified

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1 ORIGINATING ACTIVITY *(Corporate author)*

The MITRE Corporation
Bedford, Massachusetts

2a. REPORT SECURITY CLASSIFICATION

Unclassified

2b. GROUP

3 REPORT TITLE

Evaluation of ADAM, An Advanced Data Management System

4 DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

N/A

5 AUTHOR(S) *(First name, middle initial, last name)*

GILDEA, Robert A.J.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
August 1967	63	12

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
AF 19(628)-5165	ESD-TR-67-130
b. PROJECT NO. 512B	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)* MTR-442
d.	

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Deputy for Command Systems, Computer and Display Division, Electronic Systems Division, L. G. Hanscom Field Bedford, Massachusetts

13. ABSTRACT

This report evaluates the ADAM project (Advanced Data Management System), its products, applications, and some of its activities, which were part of a larger project entitled Information Systems Tools and Software Techniques. The knowledge and conclusions contained herein are intended for Air Force and other personnel who either are systems programmers or have had a brief technical orientation in information processing systems, and are interested in the management and production of software tools. There are detailed evaluations of documentation and debugging facilities, system languages and language manipulators, data structures and memory allocators. Both the design and implementation of parts of the system, as well as the entire system are discussed.

DD FORM 1 NOV 65 1473

Unclassified

Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Advanced Data Management System Information Systems Tools and Software Techniques Information Processing Systems						